Title:	Functional and Performance Assessment of Erasure Coded Storage Systems
Author(s):	Sanchez, Taylor E Sackos, Joshua P Crossman, Blair A
Intended for:	technical report
Issued:	2013-07-30

Functional and Performance Assessment of Erasure Coded Storage Systems

Taylor Sanchez - California State University San Bernardino
Josh Sackos - Washington State University
Blair Crossman - New Mexico Tech

Mentors: HB Chen and Jeff Inman

July 30, 2013

Contents

1 Abstract

Current storage systems using RAID technologies were not designed to support exascale computing systems[4]. Multiple disk drives failures (more than two) have not been an issue in the past. Conventional hard disk drive capacities continue to grow beyond the terabyte range and storage systems continue to grow to hundreds of petabytes and even exabytes. The need to efficiently handle multiple disk drive failures is now a reality and a challenging problem. Rather than separating the data from error correction or check data as parity and CRCs do, erasure codes expand the data, adding redundancy so even if a portion of the data is mangled or lost, the original data can be retrieved from the remaining portion.[4] In recent years, various research and development efforts have attempted to create an erasure code and disk-based archive storage system with power-efficiency, data integrity, and system reliability[8]. In this project, we plan to conduct function and performance assessments of erasure code storage systems. We plan to build two small test beds for an all disk parallel archive storage system and conduct tests erasure code storage software from Scality[7] and Caringo[2].

As we approach exascale computing, disk storage becomes an attractive option due to scalability of disk bandwidth over tape drive[3]. Disk failure rates make exascale archive systems prone to data loss. Replication is a possible solution, but can double or triple required storage space. Erasure coding is a promising option for an exascale archive because it offers the durability of replication with less overhead[5]. The functionality of an erasure code archive system remains untested at exascale. Our project was to build and verify the functionality of two prototype erasure storage archives using commercial products from Scality and Caringo.

Both products had the functionality to read, write, balance, and rebuild data as well as offering metadata access. Caringo did not provide us with a POSIX gateway, but has a metadata indexing tool that allowed querying. We did not have the Scality indexing tool to query the metadata, but we were provided with the POSIX interface SFUSE. The POSIX gateway caused an average bandwidth loss of 70% for small files (less than 1MB) and 50% loss for large files (greater than 1GB).

2 Functional Tests

The following system setup and functional tests were performed.

1. Setup two commercial prototype systems for:
   (a) Scality
   (b) Caringo
2. Configured erasure coding software and built two archive storage systems.
3. Used Block storage device to access Object storage
4. Conducted the following tests on both systems:
   (a) Data ingesting (write operation)
   (b) Data retrieving (read operation)
   (c) Metadata querying (index and attribute searching)
   (d) Metadata accessing
   (e) Data load balance testing
   (f) Data repairing
5. Gathered preliminary data on POSIX interface overhead

3 Testbed

The servers used in this experiment consisted of 3 SuperMicro, 6 HP Proliant , and 10 IBM servers. The nodes were interconnected a via 1GigE switch. The switch was a 3COM SuperStack3 Switch 3870 24 port model. The terminal server used was a Cyclades-TS3000 terminal server.

SuperMicro SC745

- X7DWA-N Motherboard
- Dual Intel Xeon E5410 @ 2.33Ghz LGA771Processors
- 16GB and one server has 32GB (Modules: 4GB DDR2 @ 667Mhz)
- 1 WesternDigital 1TB HDD
- 1 Hitachi 4TB HDD (only on admin nodes)
- Redundant Power Supply

HP Proliant DL160 G6

- Dual Intel Xeon E5504 @ 2.00Ghz LGA1366 Processors
- 24GB (Modules: 2GB DDR3 @ 800Mhz)
- 32GB (Modules: 8GB DDR3 @800Mhz)
- 4 WesternDigital 1TB HDD

IBM System x3755

- Quad AMD 64 Opteron
- 32GB (Modules: 2GB DDR2)
- 4 WesternDigital 1TB HDD

3.1 Scality Testbed

A SuperMicro server with a 4TB Hitachi hard drive was used as the administration node. The storage nodes in the ring consisted of 6 HP Proliant servers, each with 3 x 1TB hard drives dedicated to ring storage space. Three of the storage nodes had 24GB of ram, and the other three 32 GB. The erasure coding scheme used in the Scality ring was n=3, and k=3.

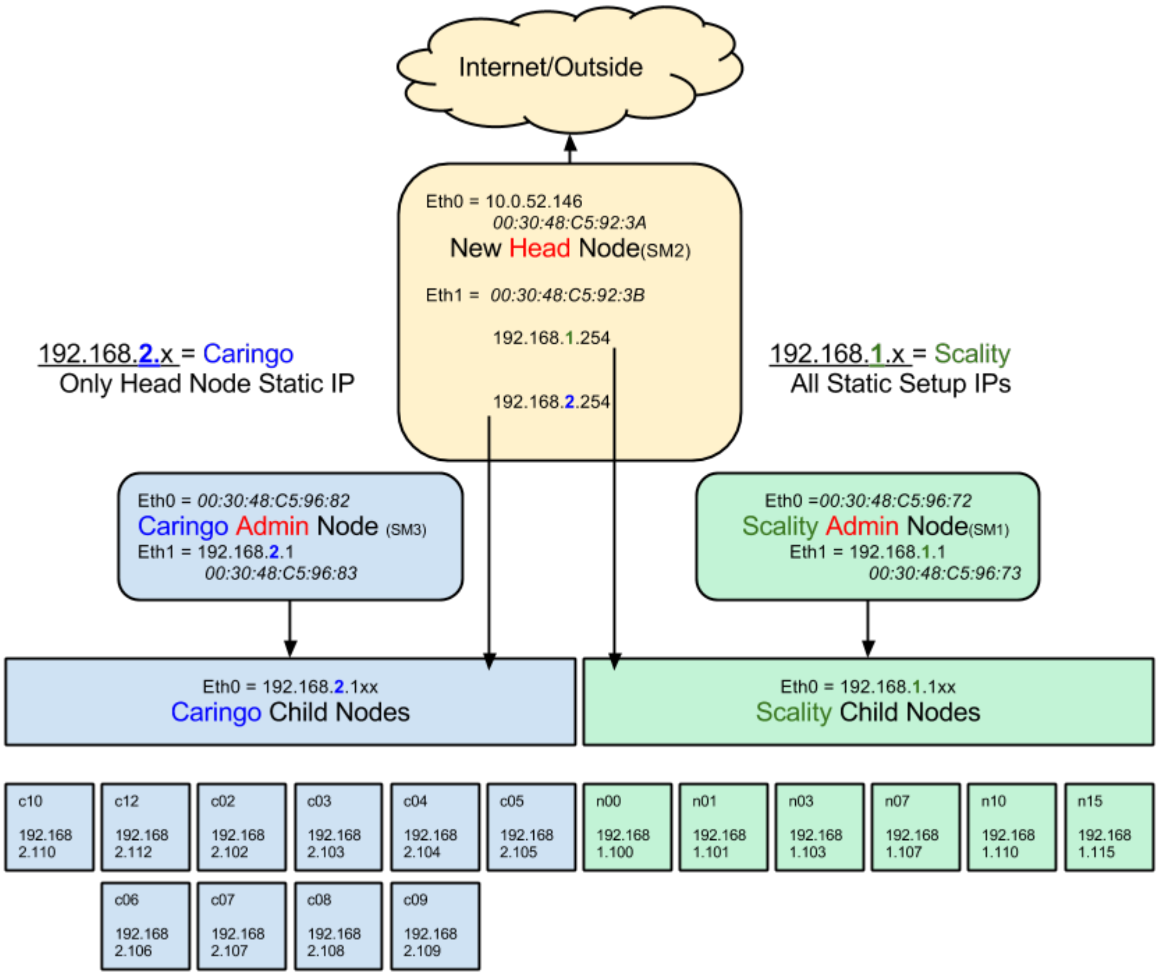
3.2 Caringo Testbed

A SuperMicro server with 32 GB of RAM and a 4TB Hitachi hard drive was used as a metadata indexer. The storage nodes in the cluster consisted of 10 IBM servers, each with 4 x 1TB hard drives dedicated to storage space. When powered on, the storage nodes PXE boot a Caringo CAStor image into RAM. The erasure coding scheme used in the Caringo cluster was n=3, and k=3.

3.3 Network Overview

Figure 1 shows a graphical overview of the testbed network. The DHCP and DNS zone configuration files are listing 3 and listing 4 in appendix A. Both Caringo and Scality make use of round robin dns forwarding to load balance data streams.

Figure 1: Network Overview.



4 Scality

4.1 Overview

For every hard drive in the Scality ring there is a daemon running that is responsible for managing its respective disk. On each Proliant node there were 6 virtual machines that ran, making a total of 36 virtual machines in the Scality ring. Virtualization allows for a slightly more flexible setup for rebuilding data. Figure 4 in appendix B shows the Scality ring [6] that was implemented in this project. Every green oval is a virtual node.

4.2 Install

To install Scality in an existing Linux system, the proper services need to be installed on each node. Once installed Scality will need to be configured, see listing 5 and 6 in appendix C for details. Further customizations will require the sproxyd.conf file to be edited, see listing 7 in appendix C for details.

4.3 The Basics: Reading and Writing

The two interfaces for the ring in this project were the SFUSE (must be installed) and REST. Currently the RESTful interface limits the maximum size of a file to upload to 500MB. This limit will be removed future releases. See listing 11 in appendix F for details on SFUSE usage and installation.

4.3.1 RESTful

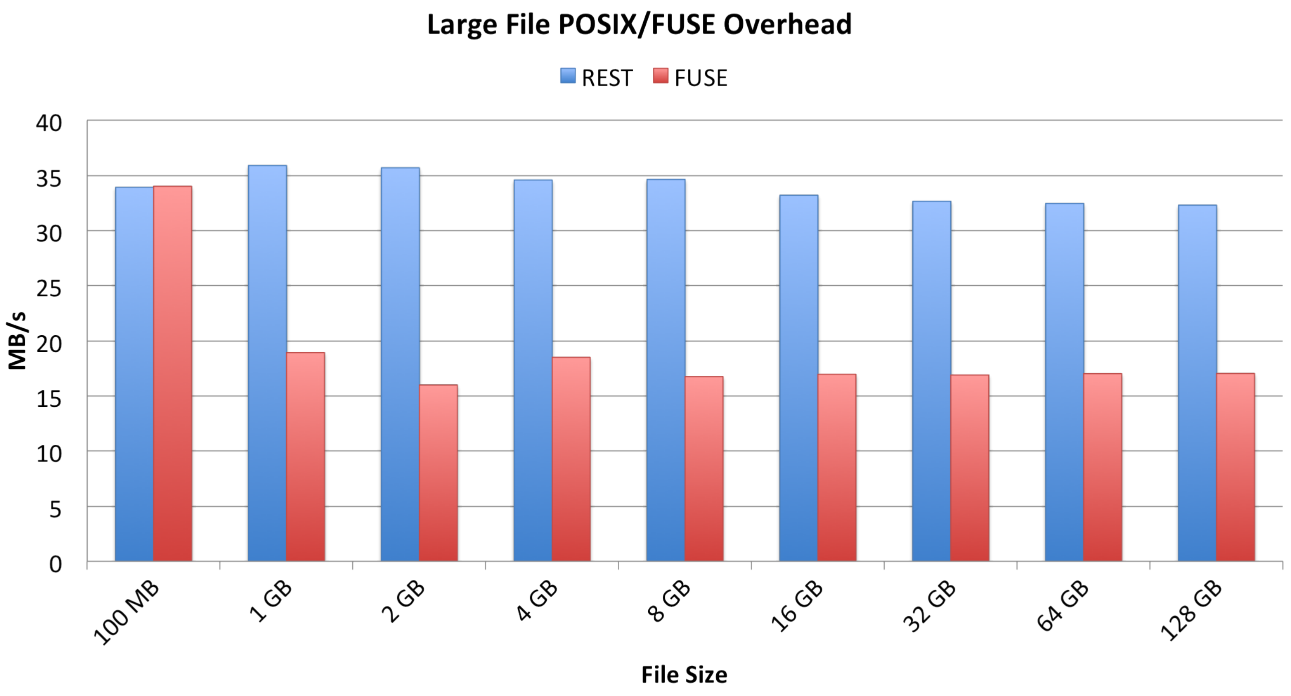
The service sproxyd will redirect your stream into the ring. Something to be aware of is that Curl loads the full file size into ram as it uploads the stream. This makes for a troublesome situation on low ram machines if Scality does remove their 500MB file upload limit. Appendix D contains a list of the curl commands.

4.3.2 POSIX

Scality provides a FUSE interface to communicate with the ring. While the POSIX interface is a requirement for the transition from tape storage to hard disk, it is less than ideal. Basic benchmarks of transferring different sized files through REST and through SFUSE (POSIX interface) interfaces were collected. In appendix F is the commands to run in bash to get SFUSE installed[6]. Once installed you should be able to use the simple cp command to put data on /ring/fuse/folder/file.name (notice it needs some sort of folder within the fuse directory, no files can go in the root of fuse). There is a large overhead with using the Scalitys POSIX interface, see figure 2 in for details.

There was not enough granularity in the tests to check where the breaking point is for the SFUSE's overhead. As you can see it does quite well on the 100MB files, but at 1GB it quickly drops. It seems reasonable to hypothesize that the 500MB data chunking size reduces the performance as SFUSE needs to start breaking up the data into chunks.

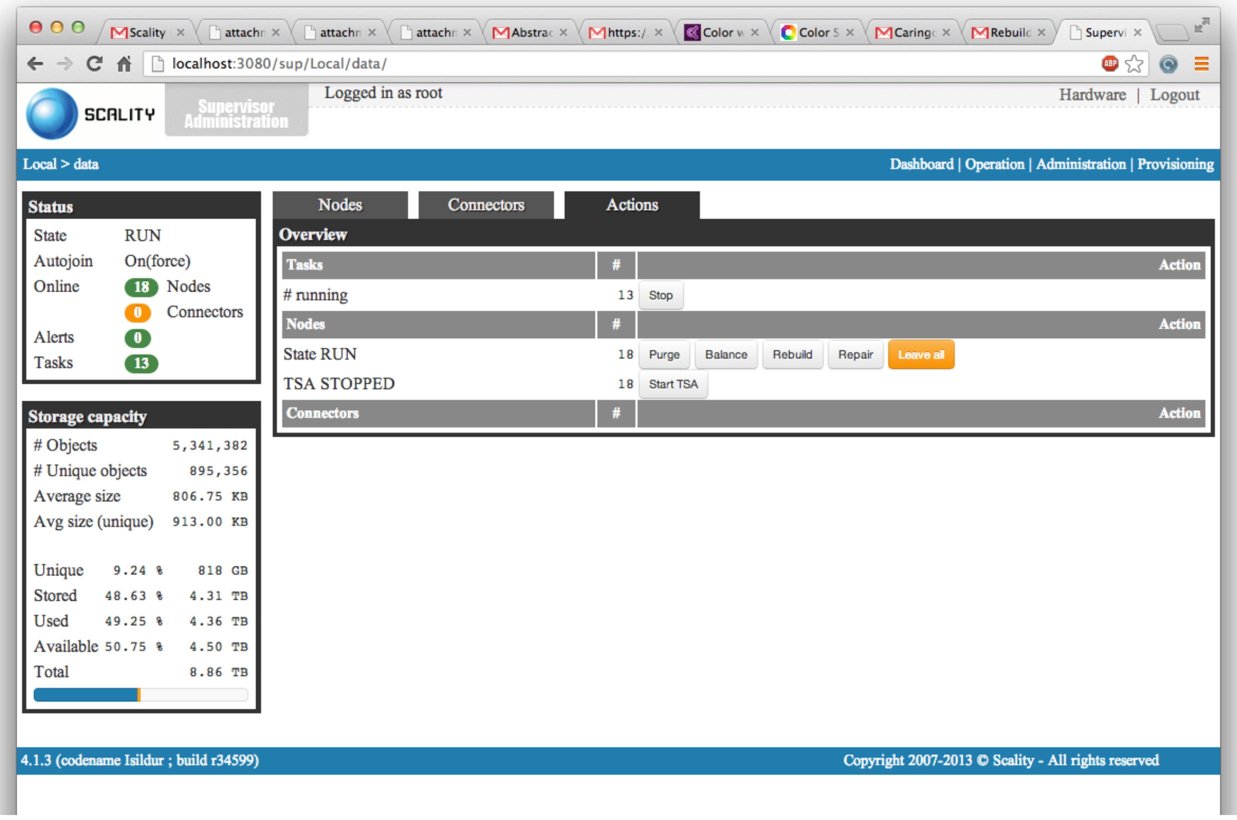
Figure 2: POSIX Overhead



4.4 Recovery Capabilities

Recovery requires a manual command to the supervisor. Data was recoverable during and after a rebuild, and when taking down servers. Performance overhead was not tested during a rebuild as fairly basic systems were used. Something worth mentioning about the supervisor is that it is not required for the ring to continue running, it can survive up to 24 hours without the supervisor available. A heartbeat fallback would probably be a good idea in a production system. Figure 3 is a screen shot where you can see the option to start the Repair action. Scality uses a type of virtual node that allows Scality to rebuild with less than required physical nodes. This means once you get the nodes back up, the ring simply needs to rebalance the data to the new nodes.

Figure 3: Scality Supervisor


localhost:3080/sup/Local/data/
SCALITY
Supervisor Administration
Hardware | Logout
Dashboard | Operation | Administration | Provisioning
Nodes
Connectors
Actions
Stop
Purge
Balance
Rebuild
Repair
Leave all
Start TSA

Logged in as root

Local > data

Status

State	RUN	
Autojoin	On(force)	
Online	18	Nodes
	0	Connectors
Alerts	0	
Tasks	13	

Storage capacity

# Objects		5,341,382
# Unique objects		895,356
Average size		806.75 KB
Avg size (unique)		913.00 KB
Unique	9.24 %	818 GB
Stored	48.63 %	4.31 TB
Used	49.25 %	4.36 TB
Available	50.75 %	4.50 TB
Total		8.86 TB

Overview

Tasks	#	Action
# running	13	Stop
Nodes	**#**	**Action**
State RUN	18	Purge Balance Rebuild Repair Leave all
TSA STOPPED	18	Start TSA
Connectors	**#**	**Action**

4.1.3 (codename Isildur ; build r34599)

Copyright 2007-2013 © Scality - All rights reserved

4.5 Data Balancing

This testing is very time consuming and was conducted throughout our testing. Although this was not the best documented we did see that the disks remained well balanced throughout our testing. The scality supervisor does list the storage usage on each node, as can be seen in the second screenshot of the Scality Overview section. One way to get a good simple look at the data balancing is to run 'df' on each node and check that they are all using a similar amount of each hard drive. The shortened output (listing 1) below was run after all testing was all finished and shows the disks to still be balanced through a rebuild, nodes being turned off for long time periods and having data written during the process.

Listing 1: Scality Balance

```
#Scality Balance
clush -w n[00,01,03,07,10,15] df 2> /dev/null | grep scality
#output:
#n00: /dev/sdb1 961432904 526904824 434528080 55% /scality/disk1
#n00: /dev/sdc1 961432904 524755448 436677456 55% /scality/disk2
#n00: /dev/sdd1 961432904 523089924 438342980 55% /scality/disk3
#n03: /dev/sdb1 961432904 527370188 434062716 55% /scality/disk1
#n03: /dev/sdc1 961432904 524312968 437119936 55% /scality/disk2
#n03: /dev/sdd1 961432904 522650476 438782428 55% /scality/disk3
#n07: /dev/sdb1 961432904 527218648 434214256 55% /scality/disk1
#...
```

4.6 Metadata

While Scality does not supply an indexing service out of the box, there are still ways of accessing the metadata contained in each server and file. The software Scality provides separately is called "Mesa". Without the Mesa software there are two ways of obtaining the metatdata in the system, one gets the metadata regarding the arc storage keys in the server, the other gets the metadata regarding the specific object.

A script to get the key-values from the servers is listed in appendix E, this produces a surprising amount of data.

The files will supply their metadata by using Curl with the corresponding simple command:

Listing 2: Single File MetaData

```
        curl −0 −I http://localhost:81/proxy/chord/fileKey
        curl −0 −I http://localhost:81/proxy/arc/fileKey
```

4.7 Scality Results and Observations

Scality didnt pass the metadata query requirement because it lacked the Mesa indexing service. Scality installs on top of an existing Linux operating system so it is very flexible and customizable. This stateful install is also inefficient as it takes up disk space that would otherwise be available to the storage cluster. The 500MB file size limit is small for the huge files that Los Alamos National Laboratory will need to push to their archive system. These problems seem possible to fix as Scality seems willing to work with their customers to improve their product. We ran a simple parallel write test on the ring to see if throughput would scale, and early testing indicates that it does.

5 Caringo

5.1 Overview

Caringo's CASTor software offers a bare-metal deployment. Each node is stateless, and can be provisioned either by usb key, or PXE boot from a network. CASTor nodes are decentralized. Each node replicates and encodes content, without any specialized nodes. When a new stream enters the CAStor cluster a "bid" is called from from the nodes in the cluster. Streams are then pushed to the node with the lowest bid, which is then responsible for encoding the stream and pushing out the data and code segments to other nodes in the cluster[2].

5.2 Install

The Caringo system installs via a RAM disk from a pxeboot setup. There is a config file that is kept on the admin node which will be delivered to the nodes during boot. When a new node comes online it needs access these files:

1. The castor file system image
2. The castor license file
3. The castor node configuration list: cfg-list.txt
4. The node configuration file pointed to by the list: node.cfg

See the examples in appendix G for how we set up DHCP for netbooting.

5.3 The Basics: Reading and Writing

Caringo's CAStor can be interfaced through either REST, or a commercially available Content File Server that allows data to be written to the cluster using a POSIX compliant NFS mount. The Content File server was not made available to us until after our testing deadline, so we were unable to access its functionality[2]. We were on the other had able to use the RESTful interface to interact with the CAStor cluster.

5.3.1 Basic Commands

Our testing was done through the RESTful interface utilizing the curl tool to create the http requests for us. Uploading objects to a CAStor cluster involves creating a bucket, and then storing objects in that bucket. The cluster will then return to you a GUID for that file, but you will still be able to access the file by simply performing a GET request on the URL that you stored the data in. One drawback of the RESTful interface is that streams are constantly moving between the nodes, so all RESTful requests must allow redirect when a 301 error is found.

See the command examples in appendix H.

5.3.2 POSIX

We received the Content File Server that serves as CAStor's POSIX interface after our testing phase had concluded[2]. We were unable to test its functionality. This interface was a requirement for the production setting as the current system is using a Block storage interface and will continue to require a Block storage interface until a transition to a RESTful backend is completed.

5.4 Recovery and Rebalance

A CAStor cluster is constantly rebuilding and rebalancing. In fact any volume or node left offline for over 14 days will be automatically reformatted and rebalanced. A supervisor may suspend rebuilding during maintenance, but the default behavior is to instantly begin a rebuild when a node goes offline. After removal of a node we found that all data could be successfully recovered, but if the number of nodes in the cluster fell beneath the total number of data and code segments, writing to the cluster would fail. For example in our cluster we had 3 + 3 erasure code. If we had 6 nodes, we could write and read from the cluster. If we fell down to 5 nodes, we could read from the cluster, but no longer write to it until the cluster again had 6 nodes. The bidding system of the nodes allows for a near seamless rebuild while writing to the cluster, as new jobs will be assigned the nodes not responsible for the rebuild. As every node in the cluster is at once a supervisor node, compute node, and storage node, there is no single point of access to the cluster that could be tied up by a rebuild. This means that as long as you have enough nodes to compute a rebuild, regular cluster management and use is seamless. [2]

5.5 Metadata

Caringo's CAStor utilizes elastic search in order to index and query metadata. The indexer will index certain basic metadata fields for any file and offers the option to increase the metadata automatically obtained with a change to the node.cfg. [1]

The metadata is easy to find, and intuitive to use. Files may be found by size, name, or any user defined metadata. See appendix I for examples.

Content can be reindexed at any time using snmp. This versatitilty comes at a price though. All streams must pass through the indexing server so that thier metadata can be indexed. If the indexing server is swamped, or runs out of memory, it can slow the storage cluster to a crawl. Therefor as the cluster increases in size, so to must the number of indexing servers increase.

5.6 Caringo Result and Observations

Caringo's CAStor is stateless. This prevents the installation from taking up a disk, but unfortunately this comes at the expense of not being able to customize the storage nodes in any way not specifically supported by the vendor.

A Caringo POSIX interface was provided past the testing deadline. Caringo promises that the Content Filer Server provides a seamless transition into the cluster from a POSIX file system mounted over NFS, and that the CFS can handle multiple writes without causing a bottleneck, but its functionality remains unverified [2].

6 Conclusions

We have found that the technology of erasure storage is a viable solution to LANLs archive system due to its scalability, parallelism, and robustness. Individually neither software meets the feature requirements. Caringo has (what appears to be) a more mature solution, but we were unable to test the

POSIX interface. The POSIX interface was one of the requirements for the archive system. Scality has a higher potential to meet the needs of the lab as they are willing to work with their customers to engineer a more fitting solution. The two biggest downfalls of Scality were that they have only beta support for single files over 500MB , and Scality currently requires a stateful (to disk) install. Continued investment into REST interfaced erasure storage has great potential to replace tape drive backups.

6.1 Futurework

There is a need to continue testing if erasure storage can meet the speed requirements of high performance computing. There will need to be some calibration done, as the erasure storage systems are currently geared towards cloud (or internet) applications. There are already plans and designated clusters to continue testing and verify the viability of erasure storage.

7 Acknowledgements

Dane Gardner - NMC Instructor
Matthew Broomfield - NMC Teaching Assistant
HB Chen - HPC-5 - Mentor
Jeff Inman - HPC-1- Mentor
Carolyn Connor - HPC-5, Deputy Director ISTI
Andree Jacobson - Computer & Information Systems Manager NMC
Josephine Olivas - Program Administrator ISTI
Los Alamos National Labs, New Mexico Consortium, and ISTI

8 References

[1] Caringo. http://www.caringo.com. `http://www.Caringo.com`.

[2] Caringo. Castor scalable, efficient and elastic object storage software. `http://www.Caringo.com/downloads/datasheets/Caringo-CAStor-Object-Storage.pdf`, June 2012.

[3] Gary Grider. ExaScale FSIO and Archive Can we get there? Can we afford to? Technical Report LA-UR-10-04611, Los Alamos National Laboratory, September 2011. Slides for presentation.

[4] J. S. Plank and C. Huang. Tutorial: Erasure coding for storage applications, part 1. Slides presented at FAST-2013: 11th Usenix Conference on File and Storage Technologies `http://web.eecs.utk.edu/~plank/plank/papers/FAST-2013-Tutorial.html`, February 2013.

[5] J. S. Plank and C. Huang. Tutorial: Erasure coding for storage applications, part 2. Slides presented at FAST-2013: 11th Usenix Conference on File and Storage Technologies `http://web.eecs.utk.edu/~plank/plank/papers/FAST-2013-Tutorial.html`, February 2013.

[6] Scality. http://www.docs.scality.com.

[7] Scality. http://www.scality.com. `http://www.Scality.com`.

[8] Marc Staimer. Cloud storage's "organic" or living evolution. Slides presented at 2011 SNIA Cloudburst Summit `http://www.snia.org/sites/default/files2/cloudburst2011/presentations/MarcStaimer_Cloud_Storage_Organic_rev02.pdf`, September 2011.

Appendices

Appendix A: Network Setup
The Admin subnet was not necessary and can be ignored.

Listing 3: dhcpd.conf

```
ddns-update-style interim;
option routers 192.168.0.254;
allow client-updates;
shared-network name{
    option domain-name "orange.com";
    option domain-name-servers newHead.orange.com;
    option subnet-mask 255.255.255.0;
#Admin Subnet
subnet 192.168.0.0 netmask 255.255.255.0{
    option subnet-mask 255.255.255.0;
    option domain-name-servers 192.168.0.254;
    option routers 192.168.0.254;
}
#Scality Subnet
subnet 192.168.1.0 netmask 255.255.255.0 {
    option subnet-mask 255.255.255.0;
    option domain-name-servers 192.168.1.254;
    option routers 192.168.1.254;
    authoritative;
}
#Caringo Subnet
subnet 192.168.2.0 netmask 255.255.255.0 {
    allow booting;
    allow bootp;
    filename "pxelinux.0";
    range 192.168.2.100 192.168.2.200;
    option subnet-mask 255.255.255.0;
    option domain-name-servers 192.168.2.254;
    option routers 192.168.2.254;
    authoritative;
}
        host caringoadm {
                option host-name caringoadm;
                option domain-name "orange.com";
                hardware ethernet 00:30:48:c5:96:83;
                fixed-address 192.168.2.1;
        }
#Scality Nodes
        host scalityadm {
                option host-name scailtyadm;
                option domain-name "orange.com";
                hardware ethernet 00:30:48:c5:96:73;
                fixed-address 192.168.1.1;
        }
        host n15 {
                option host-name n15;
                option domain-name "orange.com";
                hardware ethernet 00:26:55:18:92:50;
                fixed-address 192.168.1.115;
        }
        host n00 {
                option host-name n00;
                option domain-name "orange.com";
                hardware ethernet 00:25:b3:ae:cb:cc;
                fixed-address 192.168.1.100;
        }
        host n01 {
                option host-name n01;
                option domain-name "orange.com";
                hardware ethernet 00:26:55:18:82:5e;
                fixed-address 192.168.1.101;
        }
        host n03 {
                option host-name n03;
                option domain-name "orange.com";
                hardware ethernet 00:26:55:18:a5:02;
                fixed-address 192.168.1.103;
        }
        host n10 {
                option host-name n10;
                option domain-name "orange.com";
                hardware ethernet 00:25:b3:ae:86:48;
                fixed-address 192.168.1.110;
        }
        host n07 {
                option host-name n07;
                option domain-name "orange.com";
                hardware ethernet 00:26:55:18:a4:74;
                fixed-address 192.168.1.107;
        }
}
```

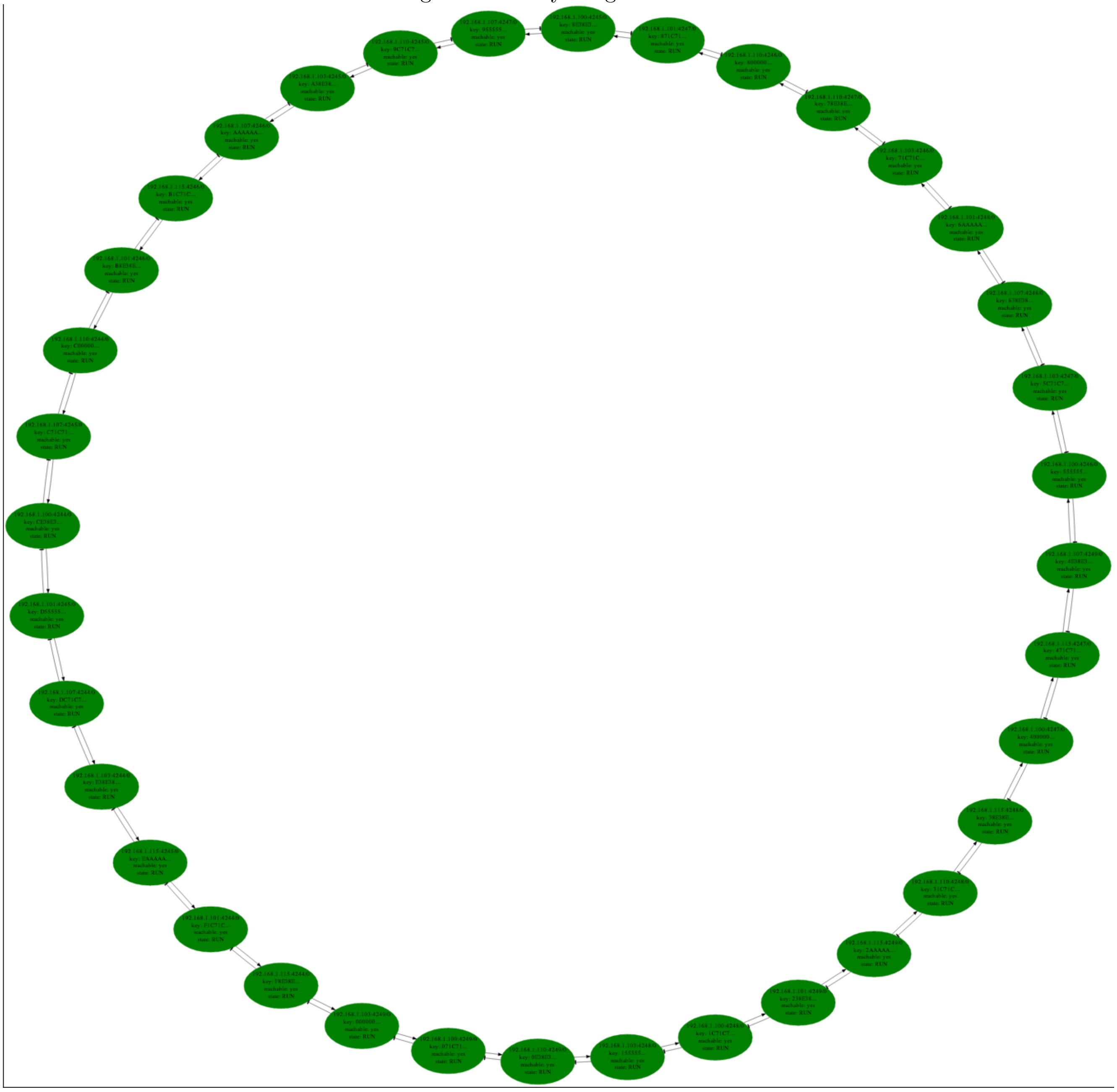
Listing 4: orange.com.fwd

```
$ORIGIN orange.com.
$TTL 1D
@ IN SOA newhead.orange.com. root.orange.com. ( ; SOA is source, so put your dns server name here, and any administrators
                                        0 ; serial
                                        1D ; refresh
                                        1H ; retry
                                        1W ; expire
                                        3H ) ; minimum
        IN NS newhead.orange.com. ; NS is loopback device
;Head/Admin subnet
newHead IN A 192.168.0.254 ; IN A is a new dns entry for the table
scalityadm IN A 192.168.1.1
caringoadm IN A 192.168.2.1

;Scality subnet
n00 IN A 192.168.1.100
n01 IN A 192.168.1.101
n03 IN A 192.168.1.103
n07 IN A 192.168.1.107
n10 IN A 192.168.1.110
n15 IN A 192.168.1.115
scality IN A 192.168.1.100
scality IN A 192.168.1.101
scality IN A 192.168.1.103
scality IN A 192.168.1.107
scality IN A 192.168.1.110
scality IN A 192.168.1.115

;Caringo subnet
caringo IN A 192.168.2.102
caringo IN A 192.168.2.103
caringo IN A 192.168.2.104
caringo IN A 192.168.2.105
caringo IN A 192.168.2.106
caringo IN A 192.168.2.107
caringo IN A 192.168.2.108
caringo IN A 192.168.2.109
caringo IN A 192.168.2.110
caringo IN A 192.168.2.112

;IPMI Interfaces
ipmi01 IN A 192.168.1.11
ipmi02 IN A 192.168.1.12
ipmi03 IN A 192.168.1.13
ipmi04 IN A 192.168.1.14
ipmi05 IN A 192.168.1.15
ipmi06 IN A 192.168.1.16
; = comment
; The origin selection will be attached to any address that does not end in a period, so remember to add final .
```

Appendix B: Scality Overview

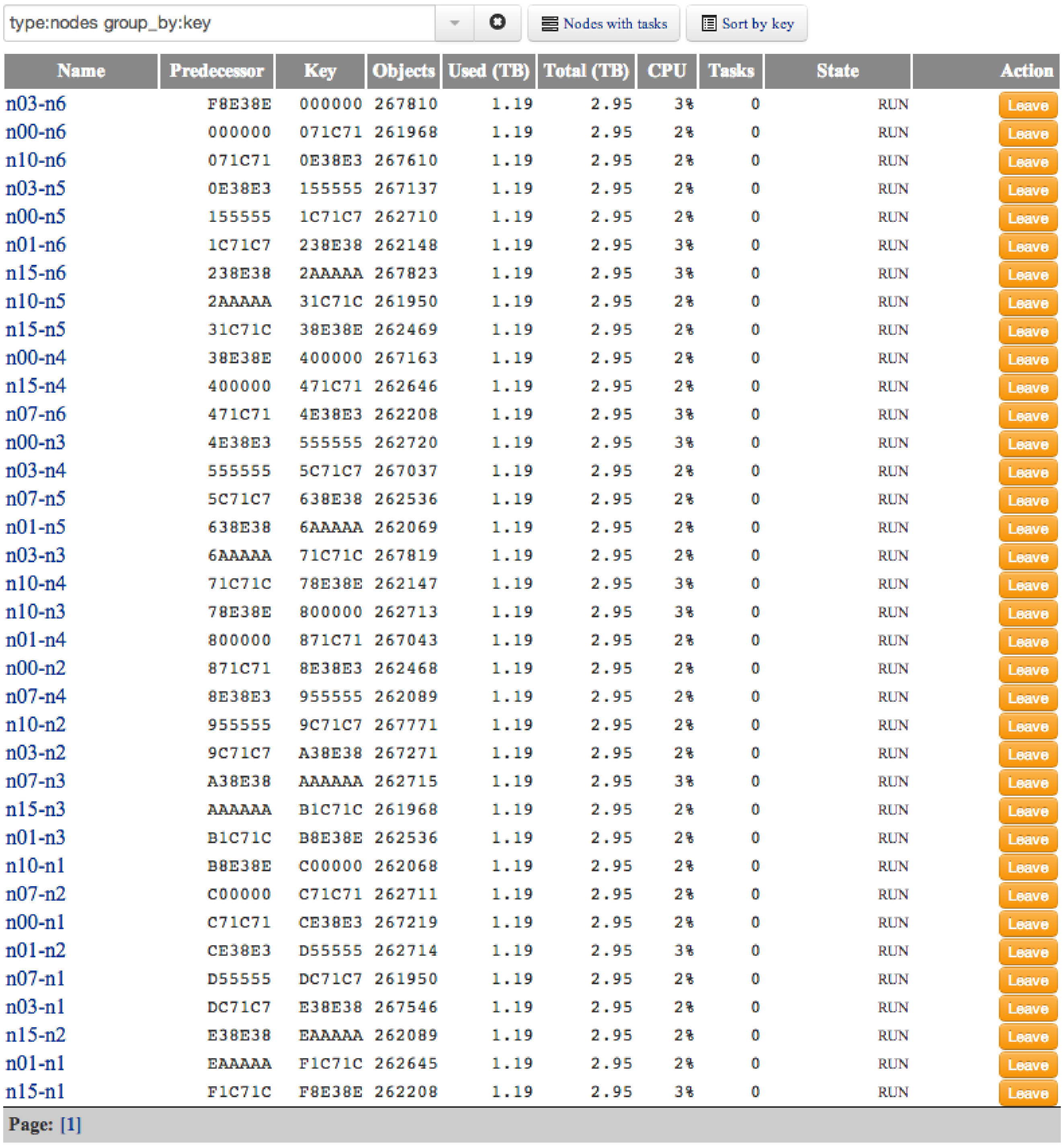
Each green oval is a virtual node.

Figure 4: Scality Ring


192.168.1.100:4245/0 key: 8E38E3... reachable: yes state: RUN
192.168.1.101:4247/0 key: 871C71... reachable: yes state: RUN
192.168.1.110:4246/0 key: 800000... reachable: yes state: RUN
192.168.1.110:4247/0 key: 78E38E... reachable: yes state: RUN
192.168.1.103:4246/0 key: 71C71C... reachable: yes state: RUN
192.168.1.101:4248/0 key: 6AAAAA... reachable: yes state: RUN
192.168.1.107:4248/0 key: 638E38... reachable: yes state: RUN
192.168.1.103:4247/0 key: 5C71C7... reachable: yes state: RUN
192.168.1.100:4246/0 key: 555555... reachable: yes state: RUN
192.168.1.107:4249/0 key: 4E38E3... reachable: yes state: RUN
192.168.1.115:4247/0 key: 471C71... reachable: yes state: RUN
192.168.1.100:4247/0 key: 400000... reachable: yes state: RUN
192.168.1.115:4248/0 key: 38E38E... reachable: yes state: RUN
192.168.1.110:4248/0 key: 31C71C... reachable: yes state: RUN
192.168.1.115:4249/0 key: 2AAAAA... reachable: yes state: RUN
192.168.1.101:4249/0 key: 238E38... reachable: yes state: RUN
192.168.1.100:4248/0 key: 1C71C7... reachable: yes state: RUN
192.168.1.103:4248/0 key: 155555... reachable: yes state: RUN
192.168.1.110:4249/0 key: 0E38E3... reachable: yes state: RUN
192.168.1.100:4249/0 key: 071C71... reachable: yes state: RUN
192.168.1.103:4249/0 key: 000000... reachable: yes state: RUN
192.168.1.115:4244/0 key: F8E38E... reachable: yes state: RUN
192.168.1.101:4244/0 key: F1C71C... reachable: yes state: RUN
192.168.1.115:4245/0 key: EAAAAA... reachable: yes state: RUN
192.168.1.103:4244/0 key: E38E38... reachable: yes state: RUN
192.168.1.107:4244/0 key: DC71C7... reachable: yes state: RUN
192.168.1.101:4245/0 key: D55555... reachable: yes state: RUN
192.168.1.100:4244/0 key: CE38E3... reachable: yes state: RUN
192.168.1.107:4245/0 key: C71C71... reachable: yes state: RUN
192.168.1.110:4244/0 key: C00000... reachable: yes state: RUN
192.168.1.101:4246/0 key: B8E38E... reachable: yes state: RUN
192.168.1.115:4246/0 key: B1C71C... reachable: yes state: RUN
192.168.1.107:4246/0 key: AAAAAA... reachable: yes state: RUN
192.168.1.103:4245/0 key: A38E38... reachable: yes state: RUN
192.168.1.110:4245/0 key: 9C71C7... reachable: yes state: RUN
192.168.1.107:4247/0 key: 955555... reachable: yes state: RUN

Each virtual node is listed below with its keys.

Figure 5: Scality Keys



type:nodes group_by:key

Name	Predecessor	Key	Objects	Used (TB)	Total (TB)	CPU	Tasks	State	Action
n03-n6	F8E38E	000000	267810	1.19	2.95	3%	0	RUN	Leave
n00-n6	000000	071C71	261968	1.19	2.95	2%	0	RUN	Leave
n10-n6	071C71	0E38E3	267610	1.19	2.95	2%	0	RUN	Leave
n03-n5	0E38E3	155555	267137	1.19	2.95	2%	0	RUN	Leave
n00-n5	155555	1C71C7	262710	1.19	2.95	2%	0	RUN	Leave
n01-n6	1C71C7	238E38	262148	1.19	2.95	3%	0	RUN	Leave
n15-n6	238E38	2AAAAA	267823	1.19	2.95	3%	0	RUN	Leave
n10-n5	2AAAAA	31C71C	261950	1.19	2.95	2%	0	RUN	Leave
n15-n5	31C71C	38E38E	262469	1.19	2.95	2%	0	RUN	Leave
n00-n4	38E38E	400000	267163	1.19	2.95	2%	0	RUN	Leave
n15-n4	400000	471C71	262646	1.19	2.95	2%	0	RUN	Leave
n07-n6	471C71	4E38E3	262208	1.19	2.95	3%	0	RUN	Leave
n00-n3	4E38E3	555555	262720	1.19	2.95	3%	0	RUN	Leave
n03-n4	555555	5C71C7	267037	1.19	2.95	2%	0	RUN	Leave
n07-n5	5C71C7	638E38	262536	1.19	2.95	2%	0	RUN	Leave
n01-n5	638E38	6AAAAA	262069	1.19	2.95	2%	0	RUN	Leave
n03-n3	6AAAAA	71C71C	267819	1.19	2.95	2%	0	RUN	Leave
n10-n4	71C71C	78E38E	262147	1.19	2.95	3%	0	RUN	Leave
n10-n3	78E38E	800000	262713	1.19	2.95	3%	0	RUN	Leave
n01-n4	800000	871C71	267043	1.19	2.95	2%	0	RUN	Leave
n00-n2	871C71	8E38E3	262468	1.19	2.95	2%	0	RUN	Leave
n07-n4	8E38E3	955555	262089	1.19	2.95	2%	0	RUN	Leave
n10-n2	955555	9C71C7	267771	1.19	2.95	2%	0	RUN	Leave
n03-n2	9C71C7	A38E38	267271	1.19	2.95	2%	0	RUN	Leave
n07-n3	A38E38	AAAAAA	262715	1.19	2.95	3%	0	RUN	Leave
n15-n3	AAAAAA	B1C71C	261968	1.19	2.95	2%	0	RUN	Leave
n01-n3	B1C71C	B8E38E	262536	1.19	2.95	2%	0	RUN	Leave
n10-n1	B8E38E	C00000	262068	1.19	2.95	2%	0	RUN	Leave
n07-n2	C00000	C71C71	262711	1.19	2.95	2%	0	RUN	Leave
n00-n1	C71C71	CE38E3	267219	1.19	2.95	2%	0	RUN	Leave
n01-n2	CE38E3	D55555	262714	1.19	2.95	3%	0	RUN	Leave
n07-n1	D55555	DC71C7	261950	1.19	2.95	2%	0	RUN	Leave
n03-n1	DC71C7	E38E38	267546	1.19	2.95	2%	0	RUN	Leave
n15-n2	E38E38	EAAAAA	262089	1.19	2.95	2%	0	RUN	Leave
n01-n1	EAAAAA	F1C71C	262645	1.19	2.95	2%	0	RUN	Leave
n15-n1	F1C71C	F8E38E	262208	1.19	2.95	3%	0	RUN	Leave

Page: [1]

Appendix C: Scality Install

C.1 Node Configuration

Listing 5: Node Config

```
yum install −y openssh−clients kpartx parted epel−release
/etc/init.d/iptables stop
/root/scality/scality−install/scripts/scality−auto−build−disks.sh −X −y −A −i sda
cp /root/scality/scality−install/config/centos6/etc/yum.repos.d/scality.repo /etc/yum.repos.d/scality.repo
echo "‘hostname −i‘ ‘hostname −f‘" >> /etc/hosts
yum −y install scality−biziod.x86_64 scality−node.x86_64 scality−ringsh.x86_64 scality−sagentd.x86_64 scality−sd−
     httpd.x86_64 scality−sfused.x86_64 scality−sfused−common.x86_64 scality−sproxyd.x86_64 scality−sproxyd−
     httpd.x86_64 scality−srebuildd.x86_64 scality−srebuildd−httpd.x86_64 scality−mod_dewpoint.x86_64 scality−
     ringsh scality−srebuildd
scality−node−config −p /scality/disk −d 3 −n 6 −u 192.168.1.1 −m ‘hostname −s‘−n −i ‘hostname −i‘
scality−sagentd−config −u 192.168.1.1
/etc/init.d/scality−node restart
/etc/init.d/scality−sagentd restart
/usr/local/scality−ringsh/config/generate−ringsh.py https://192.168.1.1:3443 data
```

C.2 Supervisor Configuration

Listing 6: Supervisor Config

```
#the 100,101,103 etc corresponded to the last 3 digits of our ip addresses for each node
for i in 100 101 103 104 107 113; do ringsh.py −r data supervisor serverAdd n$i−sagentd 192.168.1.$i 7084 ; done
echo −n "bstrap ="; ringsh.py −r data supervisor serverList | awk '{print $7}' | sed s/\:/\ /g | awk '{print $1 "
     :4244"}' | tr '\n' ','
```

C.3 Node Configuration

Listing 7: sproxyd.conf

```

#
# sproxyd config file
#

#number of worker threads per task pool
#n_workers = 100

#bind address
#bind = 0.0.0.0

#bind port
port = 10000

#connections backlog
#backlog = 10000

#number of incoming requests processed concurrently
#n_responders = 500

#maximum number of open file descriptors
max_proc_fd = 40960

#syslog_facility = "daemon"
```

```
#chroot_path =
#uid =
#gid =

#max simultaneous connections open to ring nodes
conn_max = 10000

#max number of connection re-use before kicking it
conn_max_reuse = 100000

#enable or disable consistency check on reads, when requested
consistent_reads = 1

#enable or disable consistency check on writes, when requested
consistent_writes = 1

#list of class translations to enable. It's a global parameter that
#can't be specified per interface. The list consists of 0 or more
#tokens, each token if of the form "a=b" or "a:b" (without the quotes
#and without whitespace inside the token), where a is the class stored
#in the key, and b is the interpreted class for those keys.
#class_translations = "1:2,3:4"


#
# base_path=/chord configuration
#
[chord]

#
# STRUCTURAL PARAMETERS (do not change them when set for a storage Ring)
#

#ring driver
ring_driver = "chord"

#enable path -> key conversion to use arbitrary paths as keys. a SHA1
#hash is done on the path after merging any consecutive string of '/' into
#a single '/'.
by_path_enabled = 0

#service-id of keys generated by path -> key conversion
#by_path_service_id = 0xC0

#class of service of keys generated by path -> key conversion
#by_path_cos = 2


#
# RING ACCESS PARAMETERS
#

#bootstraplist (port 4244 by default)
#eg. bstraplist = 1.2.3.4:4244,1.2.3.4:4245
bstraplist =
      192.168.1.100:4244,192.168.1.101:4244,192.168.1.103:4244,192.168.1.107:4244,192.168.1.110:4244,192.168.1.115:4244,192.168.1.



#
# CHORD PERFORMANCE TUNING
#

```

```
#enable asynchronous writes (put) triggered in request headers (see
#documentation for how to specify asynchronous behaviour in requests)
deferred_writes_enabled_by_request = 0

#enable asynchronous writes (put) for classes of service defined in
#"deferred_writes_policy" parameter
deferred_writes_enabled_by_policy = 0

#set the asynchronous writes policy per class of service. It's
#comma-separated or semicolon-separated list of class of service
#specs. Each spec is the class of service number, followed by a ':',
#followed by the minimum number of successful writes required before
#returning a success to the caller.
#deferred_writes_policy = "1:1,2:1,3:2,4:2,5:3"

#enable asynchronous deletes triggered in request headers (see
#documentation for how to specify asynchronous behaviour in requests)
deferred_deletes_enabled_by_request = 0

#enable asynchronous deletes for classes of service defined in
#"deferred_deletes_policy" parameter
deferred_deletes_enabled_by_policy = 0

#set the asynchronous deletes policy per class of service. It's
#comma-separated or semicolon-separated list of class of service
#specs. Each spec is the class of service number, followed by a ':',
#followed by the minimum number of successful deletes required before
#returning a success to the caller.
#deferred_deletes_policy = "1:1,2:1,3:2,4:2,5:3"



#
# base_path=/arc configuration
#
[arc]

#
# STRUCTURAL PARAMETERS (do not change them when set for a storage Ring)
#

#ring driver
ring_driver = "arc"

#enable path -> key conversion to use arbitrary paths as keys. a SHA1
#hash is done on the path after merging any consecutive string of '/' into
#a single '/'.
#by_path_enabled = 0
by_path_enabled = 0

#service-id of keys generated by path -> key conversion
#by_path_service_id = 0xC0

#class of service of keys generated by path -> key conversion
#by_path_cos = 0


#how many equally-spaced areas the RING is configured to support
arc_schema = 6


#
# RING ACCESS PARAMETERS
```

```
#

#bootstraplist (port 4244 by default)
#eg. bstraplist = 1.2.3.4:4244,1.2.3.4:4245
bstraplist =
    192.168.1.100:4244,192.168.1.101:4244,192.168.1.103:4244,192.168.1.107:4244,192.168.1.110:4244,192.168.1.115:4244,192.168.1.



#
# ARC STORAGE OVERHEAD CONTROL
#

#object class to use for replication
object_class = 2

#minimum size in bytes where object is not stored replicated
#(−1: always replicated)
replication_size_threshold = 60000

#maximum number of arc data chunks
n_data_parts = 3

#number of arc coding chunks
n_coding_parts = 3

#number of redundant chunks written successfully before a PUT
#is considered successful (−1: all writes must succeed)
min_redundant_parts_put_ok = 2

#minimum size in bytes of a arc data chunk
min_data_part_length = 20000


#
# ARC PERFORMANCE TUNING
#

#number of parallel tasks in the main task pool
main_n_workers = 10

#number of parallel tasks in the sub task pool
sub_n_workers = 10

#number of parallel tasks in the cache task pool
cache_n_workers = 10

#size in bytes of data stripes when cutting data on put
#stripe_size = 262144
stripe_size = 1048576

#size in bytes of data buffers read at once when reconstructing
get_reconstruct_buffer_size = 1048576

#maximum number of write buffers queued from each client before blocking
max_stripes_in_write_queue = 2

#enable or disable caching of data and metadata
#(speeds up gets that need reconstruction)
chordcache_enabled = 1



```

```
#sproxyd HOWTO using curl:

#do a PUT with base64−encoded user metadata ("myusermd")
#curl −0 −XPUT −H "x−scal−usermd: bXl1c2VybWQ=" http://localhost:81/proxy/chord/88
    FF0A8375F3112C8E340A38E38FE93438412120 −−data−binary @/etc/hosts

#do a UPDATEMD (put partial): ("newusermd")
#curl −0 −XPUT −H "x−scal−cmd: update−usermd" −H "x−scal−usermd: bmV3dXNlcm1k" http://localhost
    :81/proxy/chord/88FF0A8375F3112C8E340A38E38FE93438412120

#do a GET
#curl http://localhost:81/proxy/chord/88FF0A8375F3112C8E340A38E38FE93438412120

#do a GET with a range (retrieve 500 bytes from the byte 1000).
#ranges with at least one missing bound are not supported.
#curl −r 1000−1499 http://localhost:81/proxy/chord/88FF0A8375F3112C8E340A38E38FE93438412120

#do a STAT (dumps all HTTP headers received, user metadata included)
#curl −I http://localhost:81/proxy/chord/88FF0A8375F3112C8E340A38E38FE93438412120

#do a DELETE
#curl −XDELETE http://localhost:81/proxy/chord/88FF0A8375F3112C8E340A38E38FE93438412120


#get statistics information
#curl http://localhost:81/proxy/chord/.stats

#reset statistics information
#curl −XDELETE http://localhost:81/proxy/chord/.stats

#get active configuration parameters, each on a line with format key=value
#curl http://localhost:81/proxy/chord/.conf
```

C.4 Supervisor Configuration

Listing 8: sfused.conf

```
#
## Scality sfused configuration file
##
##

## Documentation is available here:
## http://docs.scality.com/doku.php?id=sfused:bizfs_overview#sfused

# Volume number (0..2^32)
# Scoping: when you change the 'dev' value, you don't see the objects put with another device id
dev = 987654323

# Dynamic mountpoint
mountpoint = /ring/fuse

# The logger_id string is added to the log lines to mark the logs concerning this conf.
logger_id = sfused

# check /usr/include/sys/syslog.h to see all the possibilities
# e.g.: auth, authpriv, news, syslog, mail, etc
syslog_facility = mail

conn_max = 10000
conn_tcp_nodelay = 1
conn_max_reuse = 100000

```

```
# check that a user belongs to a posix group (from /etc/group or whatever
# note that it might degrade the performance because checks are performed very often
group_check = 0

##
## Ring FS
##

file_cos = 5
cat_cos = 5
cat_page_cos = 5
dir_cos = 5
dir_page_cos = 5
rootfs_cos = 5
n_open_files_buckets = 65521
honor_forget = 1
max_proc_fd = 40960


##
## RootFS
##

rootfs = 1
rootfs_type = md5
rootfs_cache = 3
allow_rootfs_listing = 1


##
## Cache
##

# Set this property to allow a nonprivileged user to create/remove/rename directories under rootfs
allowed_rootfs_uid = 501

# Disable version checking in cache, semi-stateless mode. (recommended in failover setting)
cache_enable_checks = 0

# Force a check every x seconds for object read access, default is 0 (check all the time)
cache_check_time = 0

# Files preview (for application reading mail headers)
# To allow file previews set cache_preview_enable to "1", recommended value is disable, '0' (default value)
cache_preview_enable = 1
cache_preview_bytes = 16384 # in bytes

cache_preview_tail_trick = 1
cache_serialization_dir = /var/cache/sfused_cache
cache_serialization_period = 600

##
## Pools of worker threads
##

# Chord workers
n_workers = 480

# workers_arc_main =
# workers_arc_sub =
# workers_arc_cache =
# workers_db_index =
# workers_db_commit =
# workers_db_delete =
# workers_prefetch =

##
## Undelete
##

# Enable undelete journal to keep track of deleted keys
undelete = 1

# Purge threshold, in seconds, for undelete journal
undelete_purge_threshold = 604800

# Max age, in seconds, of deleted files in journal. Default is 604800 seconds (6 days)
# Value must be lower than the Ring purge expiration time
undelete_duration = 300

# Allow undelete for objects matching this regexp (u\.* = dovecot mails)
undelete_pattern = "^u[.].*"

# Do we match only the object's name or its full path?
undelete_pattern_full_path = 0

##
## geosync
##

#geosync = 1
#geosync_interval = 30
#geosync_prog = /usr/local/bin/ssync
#geosync_args = 'ssync $FILE $MOUNTPOINT / --dev $DEV --file-cos $FILE_COS --dir-cos $DIR_COS'


[fuse]
fuse_max_tasks = 72
direct_io = 0
big_writes = 1
read_ahead_kb = 128

[ring_driver:0]
type = chord
#bstraplist = node01-fuse.scality.com,node02-fuse.scality.com,node03-fuse.scality.com
bstraplist =
      192.168.1.100:4244,192.168.1.101:4244,192.168.1.103:4244,192.168.1.107:4244,192.168.1.110:4244,192.168.1.115:4244,192.168.1.

client_routing = 0

[ring_driver:1]
type = srest

# The base_path needs to match the path to the correct driver on sproxyd.
base_path=/proxy/arc/
#
# ## <CHANGE_ME>
#
# # NOTE: The data ring should be compromised of the first 6 node daemon processes on each storage node.
# # In this example, node daemon 1 on the first 6 servers is chosen (the port, if not listed, defaults to 4244).
# # The port used is port 81 for the httpd FastCGI module supporting sproxyd.
bstraplist =
      192.168.1.100:81,192.168.1.101:81,192.168.1.103:81,192.168.1.107:81,192.168.1.110:81,192.168.1.115:81
#
#
```

```


#Files cache

[cache:0]
ring_driver = 0
size = 2000000000
serialization = 1


#Directories cache

[cache:1]
ring_driver = 0
size = 2000000000
serialization = 1

[cache:2]
ring_driver = 0
size = 2000000000
serialization = 1

[cache:3]
ring_driver = 0
size = 100000000
serialization = 1

#File Inode Mode

[ino_mode:0]
type = mem
max_file_size = 536870912
cache = 0

#Directories Inode Mode

[ino_mode:3]
type = mem
cache = 1

#Asynchronous Inode Mode

[ino_mode:1]
type = async
max_file_size = 536870912
cache = 2
pattern = 'dovecot.index|.temp.centos6-fuse-pds.scality.com|.deleted'
queue_path = /var/cache/sfused
pattern_full_path = 0
period = 30
ttl = 60
n_workers = 12
size = 4000000000

##Sparse Inode Mode
#
[ino_mode:2]
type = sparse
cache_stripes = 0
cache_md = 0
pattern = .*
pattern_full_path = 0
sticky = 1
```

```
#
fsid = 1
main_cos = 5
page_cos = 5
stripe_cos = 5
stripe_size = 1048576
#
file_dirty_limit = 128000000 # 128MB
global_dirty_limit = 1000000000 # 1GB
dirty_timeout = 5
fsync_on_close = 1
workers_io = 64
workers_commit = 64
```

Appendix D: Scality cURL Commands

Listing 9: Scality cURL

```
#Verify "scality-sproxyd" Service is Running
#NOTE: This service needs to be running on the ring nodes, if not running then start it.
#sudo su
#service scality-sproxyd status

#Generate a Random Key
ringsh.py

#When you are in the ringsh.py program enter the following command:
key random
#Use the key that it prints to the screen in your get/put/delete curl commands.

#PUT a File
curl -XPUT -H "Expect:" -H "x-scal-usermd: bXl1c2VybWQ=" http://localhost:81/proxy/chord/70
    A249E993315C2CC3A7F10F76BCC37099447B00 --data-binary @/root/test

#GET a File
curl http://localhost:81/proxy/chord/70A249E993315C2CC3A7F10F76BCC37099447B00

#DELETE a File
curl -XDELETE http://localhost:81/proxy/chord/70A249E993315C2CC3A7F10F76BCC37099447B00


##ARC
#Generate a random ARC key, the last bit of the key is not random, but specifies settings
scalkeyarcgen -t arc -k 3 -m 3 -s 2 UPLOAD_FILE_PATH/FILE_NAME
Output: D41D8CD98F00B204E9800900000000510C302070

#PUT a File
curl -0 -XPUT -H "x-scal-usermd: bXl1c2VybWQ=" http://localhost:81/proxyd/arc/
    D41D8CD98F00B204E9800900000000510C302070 --data-binary @UPLOAD_FILE_PATH/FILE_NAME

#Get a File
curl http://localhost:81/proxyd/arc/D41D8CD98F00B204E9800900000000510C302070 >
    DOWNLOAD_FILE_PATH/FILE_NAME

#Delete a File
curl --XDELETE http://localhost:81/proxyd/arc/D41D8CD98F00B204E9800900000000510C302070

##CDMI (We didn't have these commands functioning)
```

```
curl −0 −v −XPUT −H ”Content−Type: application/cdmi−object” −H ”X−CDMI−Specification−Version: 1.0.1” −d '{ ”metadata”: { ”Data”:”bXl1c2VybWQ” }, ”value”: ”some text” }' http://jumphost:80/fs/chris80
```

Appendix E: Scality MetaData

Listing 10: ListKeys.sh

```
#!/bin/bash
DSO=$1 #data
listKeysMethod=$2 #index or browse
db=$3
table=$4
export TERM=vt100
nodes=$(ringsh.py supervisor dsoStatus $DSO | grep ”Node:” | cut −d” ” −f 2 )
now=$(date +%s)
for node in $nodes ; do
    echo ”Listing keys on $node”
    echo −e ”load conf $DSO \n\n use $node \n\n node listKeys loadmetadata=$listKeysMethod \n\n” | ringsh.py > work/keys.$node.$now
# I just redirect them from the above echo to a file, not an sql database (as the below lines would do)
# echo ”generating sql file for loading data from $node”
# sed ”s/\,/\'\,\'/g” work/keys.$node.$now | sed ”s/$/\'\)\;/g” | sed ”s/^/insert into $table values\(\'$node\'\,\'$listKeysMethod\'\,\'/” > work/keys.$node.$now.sql
# echo ”.quit” >> work/keys.$node.$now.sql
# echo ”loading data from $node in $db $table”
# cat work/keys.$node.$now.sql | sqlite3 $db
done
# xargs −t −n 1 −I{} echo −e \”load conf ring\n\n use {}\n\nnode dumpStats
```

Appendix F: Scality SFUSE

Listing 11: SFUSE Install

```
#Scality Documentation:
#http://docs.scality.com/display/DOCS/Install+the+SOFS+Connector+on+CentOS+or+RedHat

#Install Scality Package
sudo su
yum install scality−mod_dewpoint−4.1.3.r34599−1.el6.x86_64
#This may require a scality repository if I remember correctly:
#[scality−base]
#name=CentOS6 − Scality Base
#baseurl=http://scalitycs:CSrepoPass@packages.scality.com/stable_isildur/centos/6/x86_64/
#gpgcheck=0

#Copy Scality Configuration Files
cp /scality/scalityinstall/configs/scality/etc/ /etc/
scp nXX:/etc/sfused.conf /etc/sfused.conf
#Note: nXX is the DNS name or IP address of a storage node in the Scality ring.

#Create ”/ring” directory and Set Permissions
mkdir /ring
chmod g+w /ring
chown root:fuse /ring

#Mount the Control Filesystem for Fuse
```

```
mount -t fusectl none /sys/fs/fuse/connections

#Disable Transparent Hugepage Option
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled

#Install Connector
yum install scality-sfused

#Create a Catalogue
sfused -X -c /etc/sfused.conf

#Start Connector
/etc/init.d/scality-sfused start

#Test Mounted Volume
mount -l | grep fuse
```

Appendix G: Caringo Install

Listing 12: Caringo Config Files

```
################## Sample pxelinux.cfg ###############
timeout 100
default menu.c32
ONTIMEOUT 1

menu title ############# CASTor boot menu ###########
label 1
        kernel profiles/castor/kernel
        append initrd=profiles/castor/fsimage ramdisk_size=128000 root=/dev/ram0 castor_cfg=http://192.168.2.1/
             castor/cfg-list.txt

############## Sample cfg-list.txt #############
http://192.168.2.1/castor/node.cfg

############## Sample license.txt ##############
licenseFormat = 1.1

cn = Los Alamos National Laboratory
street =
l = Los Alamos
st = NM
postalCode =
co = US

clusterDescription = 1st Test Bed

# License Components
expirationDate = 2013-09-01
featureClusterMaxTB = 50
featureContentIndexing = yes
featureErasureCoding = yes
featureMinimumMinReps = 1

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.10 (GNU/Linux)

iEYEARECAAYFAlHTCbEACgkQRYikRJU1RfOrvACeLkPs9Ro5A7HKxzYc4qfyz01p
+xMAn1M8h0YbKl3QUSCRDz4fZwuWf6ND
=Vdv2
```

```
-----END PGP SIGNATURE-----

############### Sample Node configuration file ################

#to set a value the syntax is [key] = [value]
#[key] = type of setting to change
#[value] = the value to change the setting to
#a one space seperation is expected between the [key] '=' and [value]
#All headers can be chained together with dot notation.
#E.G. [cluster] \n\t name is equivalent to cluster.name


########    Cluster Name
[cluster]
        name = caringo.orange.com


########  Node Disk Formating
#Sets the nodes to use all disk volume availible
#the :k flag sets the volumes to be kept if they expire.
#Volumes are set to expire if they are not detected in the last 14 days
#the cluster was active
[disk]
        volumes = all:K


########   NTP TIME SOURCE
#sets the timesource to the local ntp server
[network]
        timeSource = 192.168.2.254
########   LOG SERVER SETUP

#Sets log host ip, the minimum error level to log, and posts object id's
#to the log file if that object has an error, instead of hiding object id's
[log]
        host = 102.168.2.1
        level = 40
        obscureUUIDs = False

########   License SETUP
#Castor requires that the product licence be serviced by a web server
[license]
        url=http://192.168.2.1/castor/license.txt


########   Security SETUP
#You may also set many security options. For our test bed we left the snmp public
#format for users is {'USER' : 'PASSWORD' }
#Instead of clear text use MD5 of following string '<username>:CAStor administrator:<password>'
[security]
        administrators = {'admin' : 'admin' , 'snmp' : 'admin'}
        operators = {'snmp':'public'}

########  ERASURE CODE SETUP
#To set the cluster to erasure code add the following lines to node.cfg
#sets encoding to 3 + 3 and minimume file size to erasure code to 60KB
[ec]
        encoding = 3:3
        minStreamSize = 61440
```

```
######## INDEXER SETUP
#The only supported indexer service is elastic search, using the
#comercial plugin provided by caringo. This set up sets elastic search
#as the indexing agent, and sets the metadata kept on each stream in the
#cluster to full
[indexer]
        name = elasticsearch
        fullMetadata = 1

[indexer_elasticsearch]
        host = HOST_IP
        port = 9200 #Default port number, can be changed
        connectionRetryInterval = 10
        insertBatchSize = 100
        insertBatchTimeout = 60
        className = caringo.castor.indexer.plugin.elasticsearchplugin.ElasticSearchPlugin
```

Appendix H: Caringo cURL Commands Cluster name = caringo.orange.com Domain name = caringo.orange.com (Doesn't have to be the same as cluster name)

Listing 13: Caringo Basic Commands

```
#BUCKET CREATION EXAMPLE
curl -i --post301 --data-binary '' --location-trusted 'http://caringo.orange.com/mybucket?domain=caringo.
    orange.com'

#PUSH example 'push up hello world'
curl -i --post301 --data-binary '<html><h1>Hello World</h1></html>' -H 'Content-type: text/html'
    --location-trusted 'http://caringo.orange.com/mybucket/helloworld?domain=caringo.orange.com'

#GET example 'get hello world'
curl -i --post301 --location-trusted 'http://caringo.orange.com/mybucket/helloworld
#GET example GUID
curl -i --post301 --location-trusted 'http://caringo.orange.com/[GUID GOES HERE]?domain=caringo.orange.
    com

#COPY example 'rename a bucket'
#change mybucket to bucket
curl -i --post301 -X COPY --data-binary '' --location-trusted 'http://caringo.orange.com/mybucket?
    domain=caringo.orange.com&newname=bucket'

#Delete a file
curl -i --post301 -X DELETE --location-trusted 'http://caringo.orange.com/mybucket/helloworld'
```

Appendix I: Caringo MetaData

Listing 14: Caringo Metadata

```
#Basic metadata
Query arg  #Description
tmBorn    #time of create or last update
size    #size in bytes
name    #UUID or name using URL encoding
content-type  #content type
etag    #entity tag
```

```
sizewithreps  #number of bytes using the maximum reps value

#Full metadata
content-base
content-disposition
content-encoding
conten-language
content-length
content-location
content-md5
last-modified
lifepoint


#Custom Fields
x-*-meta
x-*-meta-*
```

The custom fields can be filled by adding to the http header, and then queried using Get requests. Examples

Listing 15: Metadata Get Requests

```
#Query a cluster to find all the the buckets in a domain
curl -i http://caringo.orange.com?domain=caringo.orange.com&format=json

#Query a bucket for all of its contents
curl -i http://caringo.orange.com/mybucket?domain=caringo.orange.com&format=xml

#Find the amount of storage being used by a bucket
curl -i http://caringo.orange.com/mybucket?domain=caringo.orange.com&format=json&du=yes
```

Glossary

Block storage This is the underlying idea behind most file systems. The space available is blocked out to store files and folders on. 3, 9

curl Linux command used to do http request via the command line. ‘man curl‘. 6, 8

FUSE Filesystem in USerspacE. Tool to enabled Unix operating systems to create a file system in user space. A type of ”bridge” to the kernel calls. 6

Object storage Uses flexible data containers that are not set to a particular block size. The data is stored uninterpreted with metadata. 3

POSIX Portable Operating System Interface. IEEE specifications for maintaining compatibility across multiple operating system.. 3, 6, 8–10

REST REpresentational State Transfer. Makes use of the HTTP methods GET, POST, PUT and DELETE to execute operations over HTTP. This is in contrast to SOAP. 5, 6, 8, 10

SFUSE Scality’s version of FUSE to connect a user to a seeming Block storage device that is actually their ring or Object storage storage in the background. 3, 5, 6